Modeling the Evolution of Artifacts

Colette Rolland
Université de Paris 1 Panthéon Sorbonne
UFR06
17 Rue de la Sorbonne
75231 Paris Cedex 05

Abstract

The particular Requirements Engineering (RE) process modeling approach being presented in this paper advocates the capture of the history of RE artifacts. An artifact is viewed as an Evolutionary Object which evolves as the RE process proceeds. The paper proposes a classification of the various kinds of evolution of artifacts and presents a generic model, the Evolutionary Object Model, to structure the RE history kept in the artifact's memory according to this classification. It emphasises the role of RE decisions in the evolutionary process and shows how the rationale of an artifact evolution can be expressed in terms of decisions and stored in the evolutionary object history.

1: Introduction

All recent developments in the field of Software Engineering, Databases and Information Systems seem to show that support for the various stakeholders involved in Software projects can be provided by capturing the history about the design decisions in a structured manner. Much of this process knowledge is nowadays lost in the course of engineering and maintaining such systems. In large projects, in the course of time and with changing development groups, the rationale for and context of key design decisions are confused and even lost [2].

The paper aims at presenting a process tracing approach which consists of viewing any artifact as an evolutionary object which keeps track of its development history. The development activity is treated as a decision one. Thus, development artifacts evolve as a consequence of decisions. An artifact is modeled as an evolutionary object which keeps track of the decisions and the evolutions that explain its current status. The approach is based on the process model developed in the ESPRIT Basic Research Action NATURE (Novel Approaches to Theories Underlying Requirements Engineering) and implemented in the ESPRIT project F3[1] (From Fuzzy to Formal). The paper is centred on the Evolutionary Object Model. This is made in section 2. Our conclusions are drawn in section 3.

2: The Evolutionary Object Model

2.1: The Evolutionary Object

It is usually considered that a model is composed of concepts and constraints together with some definitional properties attached to them. For instance Entity-Type, Attribute, Domain, Role, and Relationship-Type are examples of concepts of the E/R model. Key and Cardinality are examples of E/R constraints. Valuation is called a definitional property of the Attribute concept (attributes can be mono/multi-valued).

Let us call each instance of a concept or a constraint of a particular model belonging to a given product under development an *object* . For instance the entity-type 'client' with the attributes 'client-number' and client-address' of the E/R schema of a customer application will be referred to as objects. The two attributes 'client-number' and 'client-address' are objects too. But the multi-valuation of the 'client-address' attribute is considered as a property of the object 'client-address'.

Objects evolve as the RE process proceeds. For instance the entity-type 'client' is first created, then refined by adding the two attributes 'client-number' and 'client-address' and finally the mono and multi valuation properties are respectively added to the definition of the two attributes. We will say that the three objects 'client', 'client-number' and 'client-address' evolve in time and we will refer to them as *evolutionary objects(EO).*

In other words we look upon a specific product, say the customer E/R schema mentioned before as a set of evolutionary objects. In fact we will see later on, that the product is rather defined as a network of evolutionary objects (an EO may be related to another EO). This view can be applied recursively to a complex product which is composed of several sub-products.

2.2: A classification of object evolution

[1] The F3 project has been partly funded by the Commission of the European Communities under the ESPRIT R&D programme. This is a collaborative project between: SEMA, France; TXT, Italy; BAe, UK; INISEL, Spain; SISU, Sweden; University of Paris 1, France, UMIST, UK; Politecnico di Milano, Italy; University of Frankfurt, Germany.

Object evolution takes different forms that we propose to classify into three categories : *transformation*, *expansion* and *mutation*. Changing the name of the entity-type 'customer' into 'client' is said to be a transformation; while adding the attributes 'client-address' and 'client-number' to the entity-type 'client' is referred to as expansion; Mapping the entity-type 'client' into the relational table 'client-table' is called mutation.

A *transformation* is a change which affects the object definitional properties; The name of an object is a property which has the particularity to be generic (every object has a name). The valuation property of an attribute is subject to transformation as well as its domain definition.

An *expansion* is a change in the object *spatial environment*. The spatial environment of an object results from the relationships existing among the concepts of the model in use. For instance, due to the definition of the relationship-type concept within the E/R model, the role 'client-order' and the role 'order-product' belong to the environment of the relationship-type 'orders'. Vice-versa these roles will participate, respectively, to the environments of the entity-types 'client' and 'product'. The attachment of attributes to entity-types implies that the 'client-number' attribute is part of the environment of the entity-type 'client' and vice-versa. A constraint associated to a concept instance belongs to its environment. Defining the 'client-number' as key of the entity-type 'client' is part of the 'client' environment. We refer to the links between objects within a spatial environment as expansion links. The structural links between concepts imply expansion links between objects.

We emphasize the notion of environment because it amplifies the semantics of an object. Considered alone, an object does not carry so much information. Placed into its environment it becomes more meaningful. Consequently changes in the object's spatial environment are part of the understanding of its history. The spatial environment of an object, say the entity-type 'client', tells us about the expansions that have occurred on the object. This information is kept through the expansion links such as the ones with 'client-number' and 'client-address' for the entity-type 'client'.

One might notice that the notion of an environment allows to propagate the information about changes. For example, as the attribute 'client-address' is part of the environment of the entity-type 'client', an evolution of the attribute, say its transformation into a multi-valued attribute, will change the client definition too.

An object *mutates* when its type changes i.e when it becomes instance of the concept *x* whereas it was instance of the concept *y*. This can occur within the same model; for instance the attribute 'client-address' could be mutated into the entity-type 'department' if it happens that the various departments of the same client can order goods autonomously and pay themselves the corresponding invoices. It can also happen between objects which are instances of concepts belonging to different models; for example, the entity-type 'client' could be further mapped into a relational table 'client-table'. These mutations will occur quite often in current system developments. Indeed, most of the time, the IS development is viewed as a process of mutation which abstracts from users requirements the conceptual specification of the system, then, converts it into a technical specification and finally into an implemented system.

We propose to introduce the notion of *temporal environment* which relates an object to the objects in which or from which it has mutated. We will say, for example, that the entity-type 'client' belongs to the temporal environment of the relational table 'client-table' and vice-versa. The link between these two objects is referred to as a mutation link because it represents a mutation between the related objects. Notice that, as the expansion links were, the mutation links are bi-directional. In the latter example one can say that the entity-type 'client' has mutated into the relational table and that the 'client-table' results of the mutation of 'client'.

Mutation links between objects come from the potential types of mutations between concepts. For instance the mutation link between the entity-type 'client' and the relational table 'client-table' results from the mapping rule stating that an entity-type is mapped into a table. The mutation links are not precisely defined nor stored nowadays. We believe that they represent an important aspect of the development process which has to be captured in the process trace.

2.3: Object evolution : a sketch

In fact our proposal consists of capturing the three kinds of object evolution and to embed in each object, in a structured way, the full history of its transformations, expansions and mutations. Each of these will be considered as a step in the RE process. The notion of environment is intensively used to structure the object history which can be seen as composed of three parts : the *inner part* which keeps track of the object transformations i.e changes in its definitional properties, the *spatial environment* which explains how the object definition has been progressively expanded, and the *temporal environment* which describes the object mutations.

Transformations, mutations and expansions that occurred in the life span of an object are actions performed as the RE process proceeds. We believe that the object history must not be limited to record that actions have been performed but should describe more about the process, for instance the contexts in which these actions occurred and their rationale as well. We think this will help in providing a useful trace in learning from it why and when actions happen and on what, to improve the way of working and reuse RE decisions in subsequent projects. In order to achieve this objective we need to define in which relevant terms the process has to be described and its trace recorded in the evolutionary object memory. In the next section we shall concentrate on the process modeling approach we propose.

2.4: Modeling the evolutionary process

Our approach is decision-oriented; this means that in our view, objects evolve as consequences of RE decisions. Decision-oriented models [4], [5] are semantically more powerful than Activity-oriented [7], [1] or Product-oriented [3] models because they explain not only how the process proceeds but also why transformations happen.

Considering the highly non-deterministic nature of the Requirements Engineering process we believe that only the last of these approaches appears to be partially appropriate. It is probably difficult to impossible to write down a realistic state-transition diagram that adequately describes what has to happen or actually happens in requirements engineering. But relying on the pure artifact history is also insufficient. Even the decision-based approaches offer only limited hints about when and how to decide on what.

Analysts react *contextually* according to the domain knowledge they acquire and react by analogy with previous situations they have been involved in. In order to take into account the very nature of the RE we have chosen to emphazise the contextual aspect of decisions. Our process modeling approach strongly couples the context of a decision to the decision itself, otherwise we loose information about the decision. This approach aims at capturing not only activities performed during the RE process but also why these activities are performed (the decisions) and when (the decision contexts).The central aspect of our process way of modeling is therefore that it makes the notion of *situation* (in which to decide) explicit and relates it to the broader question of context handling.

The four basic concepts are the ones of situation, decision, action and argument :

- *situation* to explain the decision context,
- *decision* to guide the RE process,
- *action* to perform the product transformations,
- *argument* to support decision-making.

A *situation* is most often a part of the product under development it makes sense to take a decision on. It represents the context of a decision. It can be a statement about the application domain or a requirement. It can also refer to part of existing products resulting from previous developments or a piece of generic domain knowledge. The situation may be built on either the product under development or existing products. A complete product model is presented in [8].

A *decision* reflects a choice that a requirement engineer makes at a given point in time of the RE process. A decision can be looked upon as the intention of product transformations. It explicitly represents the resolution of an issue.

An *action* performs a transformation on the product. It is a materialisation of the decision. Performing it, changes the product and may generate new situations which, in turn, are subjects of new decisions.

Arguments are statements which support or object to decisions within a given context.

The model emphasises the dimension of decisions and introduces the notion of context as a main concept

of process modelling. A *context* is a couple <situation, decision> which amplifies the decision semantics by refining the "when" part of the decision.. A more complete definition of the model can be found in [6].

Furthermore, the process model has to handle different levels of granularity of contexts, i.e. granularity of situations, as well as of decisions. This leads to the specialisation of the context into *macro* and *micro* ones. The former corresponds to a situation which requires to explore alternatives in decision making for its treatment. Refinement in decision making is modelled through the decomposition of a macro context into its more elementary context. On the other hand, when the context cannot be associated to multiple ways of solving the issue it raises then it is referred to as a micro context. Such a context implements one decision through execution of actions which transform the product under development. This raises new situations which are themselves subjects of decisions.

2.5: The evolutionary object model

The Evolutionary Object Model (EOM) is generic i.e independent of a specific product model. Our intention is to define a generic EO concept that could be adapted to various methodologies and tool environments. The concept of Evolutionary Object Class (EOC) is the unique concept provided by the EOM. The EOC is a shell from which evolutionary objects will be generated. The EOC definition is the intentional definition of the potential set of evolutionary objects.

According to our evolutionary object approach the EOC history is composed of three parts (Figure 1): the *Inner-History* which keeps track of the object transformations, the *Spatial-History* describing how the object has been expanded and the *Temporal-History* which tells us about the object mutations.

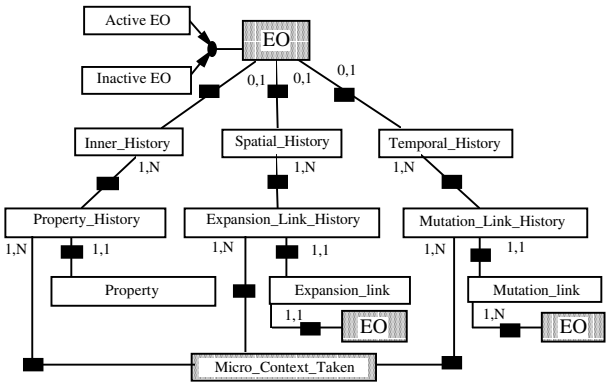

Figure 1 : EO Class : the environment aspects

Every transformation, expansion or mutation is an *Action* executed as a consequence of a micro-decision which is itself, part of a *Micro-Context-Taken.* In addition, this micro-decision can be the result of a decision making process which has required several decomposition steps of an initial macro-decision within a given *Macro-Context-Taken.* These aspects are modelled according to our process view and structured as presented in Figure 2.

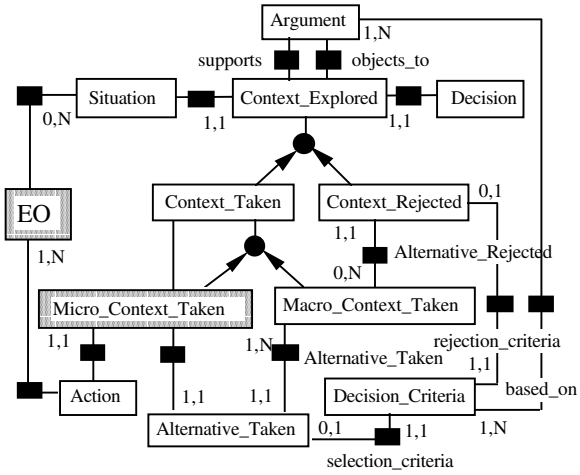

Figure 2 : EO Class : the process aspects

To summarise we can say that Figure 1 shows how the historical view of the object evolution within its spatial and temporal environments is recorded whereas Figure 2 explains how the evolutionary process providing the rationale of these evolutions is itself recorded. The former provides a structural view of the object evolution while the latter describes its dynamics.

3: Conclusion

The Evolutionary Object Model presented in this paper conforms to approaches to capturing design history.
It takes the original view of looking upon any artifact of a Requirements Engineering specification as an evolutionary object which progressively evolves as the RE process proceeds through decisions taken by the requirements engineers. The history is structured according to the classification of artifacts evolutions into transformations which affect the artifact's properties from expansions linking it to other artifacts and mutations which change its type. According to our object oriented view, the complete history of an artifact's evolution, as well as the decisions behind this evolution and their rationale are encapsulated in the object memory.

4: References

[1]Boehm B.W. : "A Spiral Model of Software Development and Enhancement"; IEEE Computer 21.
[2]Curtis B., Kasner H., Iscoe N. : "A field study of the software design process for large systems"; Comm. ACM, vol. 31, 1988.
[3]Finkelstein A., Kramer J., Goedicke M.: "ViewPoint Oriented Software Development"; Proc. Conf "Le Génie Logiciel et ses Applications", Toulouse, p 337-351, 90.
[4]Jarke M., Mylopoulos J., Schmidt J.W., Vassiliou Y.; “DAIDA - An Environment for Evolving Information Systems”; ACM Transcactions on Information Systems, Vol. 10, No. 1, 1992.
[5]Potts C.: “A Generic Model for Representing Design Methods”; Proceed. 11th International Conference on Software Engineering, 1989.
[6]Rolland C.: "Modeling the Requirements Engineering Process", Proc. Fino-Japanese Seminar on "Conceptual Modeling", 1993.
[7]Royce W.W.: "Managing the Development of Large Software Systems"; Proc. IEEE WESCON 08/1970.
[8]Schmitt J.R.: "Product Modeling in Requirements Engineering Process Modeling", IFIP TC8 Int. Conf. on "Information System Development Process", Prakash N., Rolland C., Pernici B. (eds.), North Holland (pub.), 1993